A Self-Reconfigurable Hardware Architecture for Mesh Arrays Using Single/Double Vertical Track Switches

Masaru Fukushi and Susumu Horiguchi, *Senior Member, IEEE*

***Abstract*—This paper deals with the issue of reconfiguring mesh-connected processor arrays (mesh arrays) in the presence of faulty processors. For massively parallel systems, it has become necessary to develop built-in self-reconfigurable systems that can automatically reconfigure partially faulty systems. Many reconfiguration methods have been proposed to date; however, most of them are not suitable for self-reconfiguration. In this paper, we propose a self-reconfiguration method based on simple column bypass and south directional rerouting schemes. This proposal offers the combined advantages of high probability of successful reconfiguration, low hardware overhead, and simplicity of implementation. A switching mechanism, which can determine the desired switch functions automatically using the states of neighboring processors, makes the implementation of our method easier. Simulated results show that the proposed method achieves a higher system yield than that of previous methods with half the number of redundant switches and interconnections. The prototype system of self-reconfigurable mesh arrays is implemented using a field-programmable gate array (FPGA) and the hardware overhead is discussed.**

***Index Terms*—Fault tolerance, field-programmable gate array (FPGA), self-reconfiguration, very large scale integration (VLSI)/wafer scale integration (WSI) mesh array, yield enhancement.**

I. Introduction

THE TWO-DIMENSIONAL mesh-connected processor array (mesh array for short) is one of the most popular architectures used in parallel processing. Some computational processes, such as matrix manipulation, image processing, and pattern recognition, are known to be well matched to this topology. Examples of mesh connected systems include the Illiac IV, Goodyear Aerospace MPP, Touchstone Delta, and Intel Paragon [1]. Because of the regularity of the mesh structure, it is suitable for efficient very large scale integration (VLSI) or wafer scale integration (WSI) implementations, in which all processing elements (PEs) and the interconnections among them can be integrated into single or multiple chips or wafers. Using these implementation methodologies, high-speed, low-power, massively parallel mesh arrays can be realized in an extremely small area.

However, one major obstacle to realizing such large-scale mesh arrays is an efficient reconfiguration mechanism to enhance system yield and reliability. Manufacturing defects are inevitable during fabrication, and the yield of a system is usually very low. It is almost impossible to guarantee that all the PEs in the system will run without faults throughout its working lifetime. In such cases, instead of regarding the whole system as faulty, it is cost efficient to reconfigure the mesh connection by incorporating additional hardware such as spare PEs, switches, and redundant interconnections.

So far, extensive research has been dedicated to reconfiguring a faulty physical mesh array into a nonfaulty logical mesh array [2]–[24]. A particularly simple, but very useful, architecture is the 1 1/2 track switch model (1 1/2-TS model for short) proposed by Kung [2]. This model consists of an $N \times N$ mesh array, a single row (or column) of spare PEs along each boundary, and single-track switches which are placed between every two adjacent rows (columns) of PEs to avoid faulty PEs. In this model, an $N \times N$ array is realized by assigning spare PEs to all faulty PEs and the combination is searched exhaustively by graph theory. Since the 1 1/2-TS model has the advantage that the hardware overhead is small and physical distances between logically adjacent PEs are bounded by a small constant, many reconfiguration methods have been proposed based on this model [7], [8], [14], [16], [17]. Some other models, [3], [5], [6], [13], [15], [19], use multiple-track switches to enhance the yield/reliability of mesh arrays. For example, the three-track-one-spare model [6] requires three-track switches between every two adjacent rows (columns) of PEs to achieve higher reconfiguration capability than the 1 1/2-TS model. The two-track design of [24] reduces the number of switches and tracks by using more complex switches, while keeping the same reconfiguration capability as the three-track-one-spare model. The models of [12] and [20] employ a row and column elimination technique to reduce hardware overhead. A variety of reconfiguration methods are also presented in the literature, e.g., in [4], [9]–[11], [18], [21]–[23].

In today's VLSI circuit technologies, as the number of PEs integrated in chips/wafers has greatly increased, it has become necessary to develop built-in self-reconfiguration mechanisms which can automatically reconfigure a partially faulty array [20]. Although some of the above methods achieve a high probability of successful reconfiguration using global information on fault distributions, they are too complex to be practically implemented in hardware. To be considered viable for the purpose of built-in self-reconfiguration, a reconfiguration method must satisfy two challenging and mutually conflicting design requirements. First, it must lend itself to easy implementation using very little hardware, and, second, the method must have

Manuscript received May 1, 2003; revised November 16, 2003. This work was supported in part by the Grant-in-Aid of Scientific Research (B)14380138, Japan Science Promotion Society.

The authors are with the Graduate School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, 923-1292, Japan.

Digital Object Identifier 10.1109/TIM.2003.822717

the capability of reconfiguring efficiently at high execution speed. As well as reconfiguration in fabrication time, such self-reconfiguration is essential in run-time to minimize the time overhead for real-time and mission-critical applications.

Numata and Horiguchi [7], [8] proposed a bypass and shift (BS) method to achieve self-reconfiguration using only local fault information from neighboring PEs. The BS method uses recursive signals to avoid deadlock, making the hardware implementation more difficult. Smith [20] proposed a self-repairable memory array based on the row and column bypass technique. However, this method is not efficient for processor arrays in which each element has more complicated functions than those of a memory cell. Takanami *et al.* proposed a neural method [14] and an approximate method [16], [17] for the 1 1/2-TS model to find the assignment of spare PEs. Although these methods achieve the same yields as the exhaustive method [2] by approximation, the hardware overhead for additional switches and tracks is double that of our architecture.

In this paper, we propose a self-reconfiguration method for mesh-connected processor arrays and show a prototype system of self-reconfigurable mesh arrays implemented on a field-programmable gate array (FPGA). In contrast to existing self-reconfiguration methods, our method offers the combined advantages of high probability of successful reconfiguration, low hardware overhead, and simplicity of implementation.

The rest of this paper is organized as follows. Section II describes in detail the architecture of a self-reconfigurable mesh array. Section III proposes a reconfiguration method employing simple column bypass and south directional rerouting schemes. The reconfiguration performance of the proposed method is compared with previous studies in Section IV. A prototype system of self-reconfigurable mesh arrays is implemented on an FPGA and the hardware complexity is discussed in Section V. Finally, Section VI concludes the paper.

II. Architecture

The objective of mesh array reconfiguration is to obtain a fault-free array from a faulty array by avoiding faulty PEs. In this paper, a physical array is defined as an array which may include some faulty PEs, and a logical array is defined as a fault-free array after reconfiguration.

The architecture of our proposed physical array is shown in Figs. 1 and 2. Fig. 1 illustrates the layout of PEs, switches, tracks, links, and the connection between them. $(M-m)$ rows and $(N-n)$ columns of PEs are designated as spare and are identical to nonspare PEs in terms of function and performance. In Fig. 1, the spare PEs are illustrated at the bottom two rows and right two columns; however, the distribution is arbitrary in our architecture since all PEs are treated uniformly during the reconfiguration processes. As a result of reconfiguration, some PEs on the bypassed columns and bottom rows are designated as spare PEs.

Each switch has four functions, east-west (*EW*), north-west (*NW*), north-east (*NE*), and no connection (*NC*), as shown in Fig. 1. The initial function for all switches is *EW*. Port selectors in Fig. 1 select m I/O ports among possible M ports on both the right most and the left most sides of the array, and n ports among N ports on the upper most and the lower most sides in order to

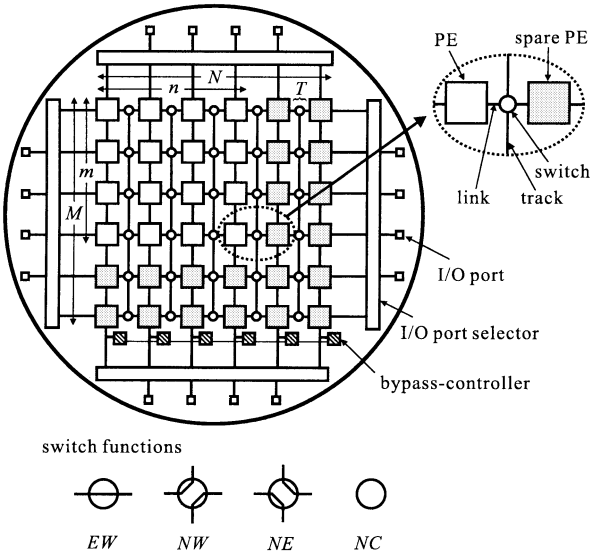


Fig. 1. $M-N-m-n-T$ type architecture.

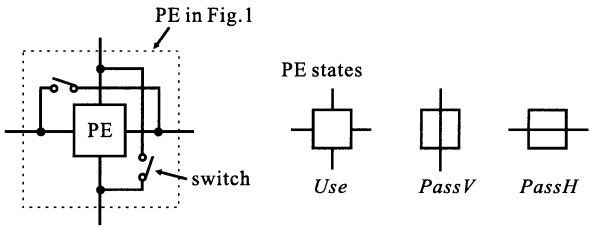


Fig. 2. Internal structure of a PE.

connect with external I/O ports. A bypass controller is allocated at the bottom of every column to control and perform column bypass (explained in the next subsection). We refer to such self-reconfigurable architecture as "$M-N-m-n-T$" type. In terms of the layout of switches and tracks, this architecture is the same as that described in [7] and the hardware overhead is half that of the 1 1/2-TS model, which has switches and tracks between every two adjacent rows/columns.

Each PE in Fig. 1 has two switches, as shown in Fig. 2, so that it can be bypassed and converted into a connecting element. Using these switches, each PE has three possible states; *Use*, pass vertically (*PassV*), and pass horizontally (*PassH*). The *PassV* and *PassH* states correspond to bypassing the PE vertically and horizontally, respectively. The *PassH* state is used for bypassed PEs and the *PassV* state is for other faulty PEs. Since the switches only have two states, on or off, the hardware structure of the switch is much simpler than that shown in Fig. 1. It is assumed that each PE outputs a signal indicating whether the PE is faulty or not. This function can be realized by the built-in-self-test (BIST) circuit, though discussion of this hardware circuit is outside the scope of this paper.

Obviously, the power of the reconfigurable architecture is determined by the total amount of available hardware resources, such as switches and tracks, and their distribution in the array. Although one would like to use as much hardware as possible to increase reconfiguration capability, it is often expensive to do so because the probability of faults on the redundant hardware

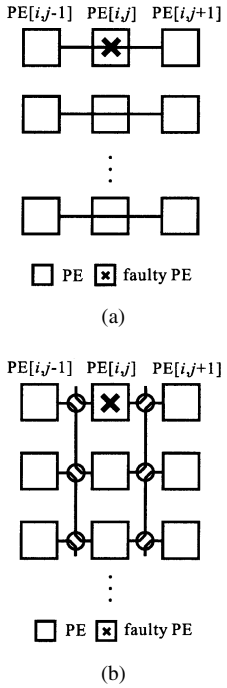


Fig. 3. Two basic schemes to avoid faulty PEs. (a) Column bypass. (b) South directional rerouting.

will increase and may decrease the system yield. From the viewpoint of hardware overhead, we concentrate in this paper on the architecture of single or double tracks ($T = 1$ or 2). In this architecture, we pay attention to replacing only faulty PEs, because switches, tracks, and control circuits use much less hardware as compared to PEs, and the probability of their being faulty is much less.

The physical addresses for all PEs and switches are defined as follows.

Definition 1: Each PE is addressed from the upper left and the PE on the ith row and the jth column is denoted as PE$[i, j]$ $(1 \leq i \leq M, 1 \leq j \leq N)$. The indices i and j are called the row index and column index, respectively. Each switch is also addressed in the same way and denoted as SW$_T[i, j]$ $(1 \leq i \leq M, 1 \leq j \leq N - 1, T = 1, 2)$. Here, SW$_2[i, j]$ is allocated on the right side of SW$_1[i, j]$.

A. Basic Schemes

Two simple schemes, column bypass and south directional rerouting, are employed to avoid faulty PEs. In the column bypass scheme, one column is removed from a physical array and the two columns adjacent to the bypassed column are connected directly. Fig. 3(a) presents an example of column bypass in which the jth column is bypassed and $(j - 1)$th column and $(j + 1)$th column are connected with each other. These connections are achieved by changing all PE states in the bypassed column into *PassH*; hence, no external switch is required in this scheme. In the rerouting scheme, a faulty PE is removed from a physical array in such a way that the faulty PE and possibly some other nonfaulty PEs reroute their interconnections using switches and tracks. Fig. 3(b) shows an example of south directional rerouting in which faulty PE$[i, j]$ and PE$[i', j]$s in the south direction $(i < i')$ are rerouted to avoid the faulty PE. Note that interconnections among PEs are limited by the number of tracks T, that is, PE$[i, j]$ can connect with either PE$[i - T, j + 1], \cdots,$ PE$[i, j + 1], \cdots$, or PE$[i + 1, j + 1]$. North directional rerouting is defined in the same way. If external switches and tracks are also placed between every consecutive row, as in the 1 1/2-TS model, east and west directional rerouting can also be defined in this way.

Clearly, the rerouting scheme, which requires switches and tracks, is more effective than the bypass scheme for avoiding faulty PEs. If many switches and tracks are employed and if rerouting is allowed to any direction as in [6] and [24], a high probability of successful reconfiguration can be achieved. However, such reconfiguration processes become much more complicated and ultimately inappropriate for self-reconfiguration. Therefore, only one-directional (south) rerouting and column bypass are employed in our architecture for easy hardware implementation and lower hardware overhead.

III. Reconfiguration Method

A. Outline of Proposed Method

The proposed reconfiguration method for $M-N-m-n-T$ type architecture is described in this section. The method consists of the following three steps in order to obtain a fault-free $m \times n$ logical array from a faulty $M \times N$ physical array.

Step 1) Bypass $(N - n)$ columns in order of faulty PE number.
Step 2) Deactivate all PEs which do not have proper interconnections.
Step 3) Change all switch functions simultaneously.

These three steps are described in detail in the following subsections. In our method, south directional rerouting is performed in such a way that each switch determines its desired function automatically using only the states of neighboring PEs. The proposed reconfiguration method is characterized by automatically bypassing columns and changing switch functions, which is why we have called it the bypass and change (BC) method.

B. Bypassing Columns

Step 1 of the BC method is to bypass $(N - n)$ columns from an $M \times N$ physical array. As reported in [7], bypass is an efficient strategy for clustered fault models where a large number of faults are concentrated on parts of a wafer. After bypassing $(N - n)$ columns, the size of $M \times N$ physical array is reduced to $M \times n$. The following variables are defined to describe our bypass algorithm.

Definition 2: Fault signal from the PE$[i, j]$ is denoted as fault$[i, j]$, which has a value of either 0 (fault-free) or 1 (faulty).

Definition 3: $f_{\text{in}}[i, j]$ and $f_{\text{out}}[i, j]$ are the input value and the output value of PE$[i, j]$, respectively, and are defined as follows:

$$f_{\text{in}}[i, j] = \begin{cases} 0 & (i = 1), \\ \sum_{k=1}^{i-1} \text{fault}[k, j] & (2 \leq i \leq M) \end{cases} \quad (1)$$

$$f_{\text{out}}[i, j] = f_{\text{in}}[i, j] + \text{fault}[i, j]. \quad (2)$$

It is obvious that $f_{\text{out}}[M, j]$ corresponds to the total number of faulty PEs on the jth column.

```
COL := {1, ···, N};
f_out[M, l(0)] := 0, f_out[M, r(N)] := 0;
Num := N − n;
a_max = f_max := 0;

for all j ∈ COL do
  for i=2 to M do
    f_in[i, j] = f_out[i − 1, j];
  end for
end for

while Num > 0 do
  for all j ∈ COL do
    a[j] := f_out[M, l(j)]+f_out[M, j] +f_out[M, r(j)];
  end for

  /* the first search */
  for j=1 to N such as j ∈ COL do
    if (f_out[M, j] > f_max) or
      (f_out[M, j] − f_max and a[j] > a_max) then
      f_max := f_out[M, j], a_max := a[j];
    end if
  end for

  /* the second search */
  for j=1 to N such as j ∈ COL do
    if f_out[M, j] = f_max and a[j] = a_max then
      for all i=1 to M do
        SW_T[i, j] := EW, PE[i, j] := PassH; /* bypass */
      end for
      f_max := −1;
      Num := Num − 1;
      COL := COL − {j};
    end if
  end for
end while
```

Fig. 4. Bypass algorithm.

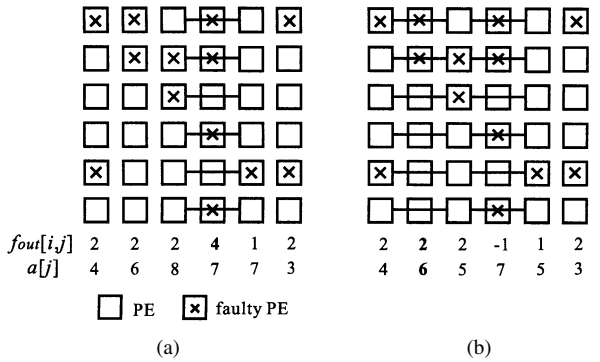


Fig. 5. Example of column bypass. (a) The fourth column is bypassed. (b) The second column is bypassed.

Definition 4: $r(j)$ and $l(j)$ are defined as the column indices of the logically adjacent right and left columns of the jth column, respectively, where $l(j) < j < r(j)$.

Definition 5: COL is the set of indices of nonbypassed columns.

Fig. 4 shows the algorithm which determines the columns to be bypassed. In this algorithm, a search from the leftmost column to the rightmost column is conducted twice in order to bypass one column. In the first search, the maximum number of total faulty PEs in each column is obtained (denoted by $f_{\max}$ in Fig. 4). In the second search, the column whose $f_{\text{out}}[M, j]$ is equal to the $f_{\max}$ is bypassed. Note that all functions of $\text{SW}_1[i, j]$ and $\text{SW}_2[i, j]$ become *EW* when the jth column is bypassed. In order to bypass a column in the area where the most faulty PEs are clustered, both $f_{\text{out}}[M, j]$ and $a[j]$ have to be taken into consideration as shown in Fig. 4. In this sense, the bypass algorithm is superior to that of [7]. The example of column bypass is demonstrated in Fig. 5, where the values of $f_{\text{out}}[M, j]$ and $a[j]$ are also shown under every columns. In the first trial of bypassing, the fourth column from the left is bypassed as shown in Fig. 5(a) and then the second column is bypassed as shown in Fig. 5(b). In Fig. 5(b), the bypass algorithm of [7] will bypass the rightmost column and it may cause a lack of spare PEs in the third column when the number of spare PEs, $M - m$, is equal to 2.

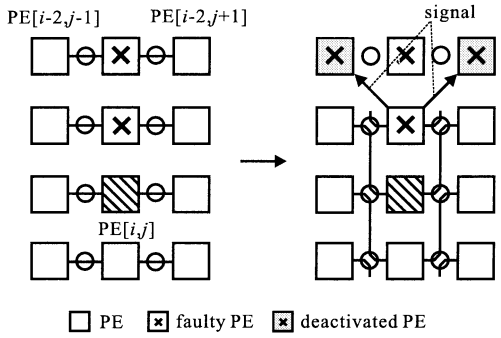


Fig. 6. Deactivation by sending signals.

C. Deactivating PEs

Faulty PEs should be deactivated and replaced by other fault-free PEs; however, not all fault-free PEs are used in the logical array. Due to the connection limitations mentioned in Section II, there may be PEs that have no proper interconnections, as shown in Fig. 6. In this figure, PE$[i-2, j-1]$ and PE$[i-2, j+1]$ do not have proper interconnections since PE$[i, j]$ can connect only with either PE$[i-1, j+1]$, PE$[i, j+1]$ or PE$[i+1, j+1]$ when T is equal to 1. As long as such PEs continue to be active, a logical array will contain broken rows; therefore, these PEs should be deactivated. In this case PE$[i-1, j]$, which detects the existence of such PEs, sends a signal toward both PEs to deactivate them. Some variables are defined to describe the deactivation more precisely.

Definition 6: PE$[i, j]$ holds a variable $\text{deact}[i, j]$ which has a value of 0 (not deactivated) or 1 (deactivated).

Definition 7: PE$[i, j]$ also holds a variable $\text{unused}[i, j]$ which is defined as $\text{fault}[i, j]$ or $\text{deact}[i, j]$.

Definition 8: $F_{\text{in}}[i, j]$ and $F_{\text{out}}[i, j]$ are defined by replacing $\text{fault}[i, j]$ with $\text{unused}[i, j]$ in (1) and (2) in Definition 3 to count the total number of PEs that cannot be used in a logical array.

Fig. 7 shows the algorithm for deactivating PEs, in which three tasks, count $F_{\text{out}}[i, j]$, transfer signals, and decision of deactivation, are performed in all PEs in parallel. Note that the PEs which should be deactivated are decided step by step; that is, a deactivated PE may begin to send a signal and cause other PEs to be deactivated. In order to deactivate PEs completely, the above three tasks are repeated at most $M \times N$ times, and the BC method has to wait during the process. An example of deactivation steps is shown in Fig. 8. Fig. 8(a) shows the situation after the column bypass shown in Fig. 5. The number on each PE in Fig. 8 corresponds to $F_{\text{in}}[i, j]$. Deactivation of the PEs is done in parallel, as shown in Fig. 8(b), and some deactivated PEs may be reactivated due to the change of the $F_{\text{in}}[i, j]$, as shown in Fig. 8(d). In this example, all information on $F_{\text{in}}[i, j]$ and $\text{unused}[i, j]$ remain unchanged even if the time steps continue to be increased.

```
step := 0;
for all 1 ≤ i ≤ M and j ∈ COL do
  while step ≤ M × N do
    /* (1) count F_out[i, j] */
    F_in[i + 1, j] = F_out[i, j];
    /* (2) transfer signals */
    if fault[i, j] = 1 and F_in[i, j] > T then
      Transfer signals to both PE[i − T, l(j)] and PE[i − T, r(j)];
    end if
    /* (3) decision of deactivation */
    if receive signal from PE[i + T, r(j)] then
      if F_in[i + T, r(j)] − F_in[i, j] ≥ T then
        flag_r := 1; else flag_r := 0;
      end if
    end if
    if receive signal from PE[i + T, l(j)] then
      if F_in[i + T, l(j)] − F_in[i, j] ≥ T then
        flag_l := 1; else flag_l := 0;
      end if
    end if
    deact[i, j] := flag_r OR flag_l;
    step := step + 1;
  end while
end for
```

Fig. 7. Algorithm for deactivating PEs.

Evaluation of this technique is difficult because the exact number of time steps required for deactivation depends on the number of faults and their distribution. Here, we consider the worst case.

Lemma 1: The deactivation takes at most $M \times N$ time steps for complete deactivation.

Proof: Consider the case that all PEs are deactivated by signals from one PE. Although such a case never occurs because the deactivation is performed in all PEs in parallel, in this case, it takes $M \times N$ time steps. For this reason, step 2 of the BC method waits just $M \times N$ time steps. ■

Lemma 2: After deactivation of PEs, the value of $F_{\text{in}}[i,j] - F_{\text{in}}[i,r(j)]$ between two adjacent PEs on nonbypassed columns is between $-T$ and T.

Proof: This follows directly from the conditions in (3) in the algorithm shown in Fig. 7. ■

Lemma 2 indicates that each PE$[i,j]$ on a nonbypassed column can find a PE on the $r(j)$th column to connect to which is fault-free and nondeactivated. Since just m PEs in each nonbypassed column are used to construct a logical array, such fault-free PEs are also deactivated at the end of Step 2 of the BC method that satisfy the condition $i - F_{\text{in}}[i,j] > m$.

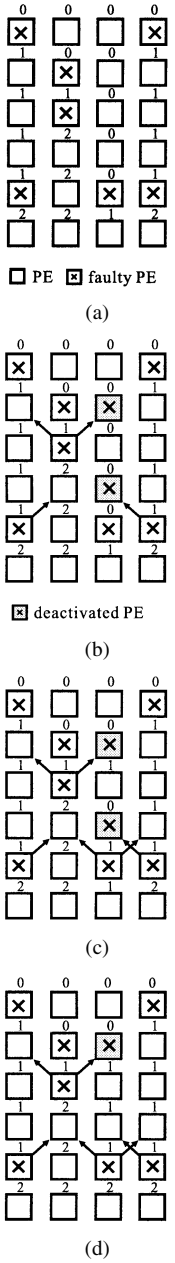


Fig. 8. Example of deactivation steps. (a) Step = 0. (b) Step = 1. (c) Step = 2. (d) Step = 3.

D. Changing Switch Functions

After unused PEs have been bypassed and deactivated, all switch functions can be simultaneously changed from their initial function *EW* to the desired function automatically. Fig. 9 shows two proposed rules to change a switch function for $T = 1$ and $T = 2$. These rules require only local information to be held in neighboring PEs. In the rule for $T = 1$, for example, if $F_{\text{in}}[i,j] - F_{\text{in}}[i,r(j)] = 1$, then the function is determined by *NW* without referring to $\text{unused}[i,j]$ and $\text{unused}[i,r(j)]$; if $F_{\text{in}}[i,j] - F_{\text{in}}[i,r(j)] = 0$ and both PE$[i,j]$ and PE$[i,r(j)]$ are fault-free, then the desired switch function is determined by *EW*. From Lemma 2, it can easily be seen that it is not necessary to define switch functions for the cases of $|F_{\text{in}}[i,j] - F_{\text{in}}[i,r(j)]| > T$. The rule for $T = 2$ requires more fault information than that for $T = 1$, namely $\text{unused}[i-1,j]$ and $\text{unused}[i-1,r(j)]$. Fig. 10 illustrates reconfiguration examples of the $M-N-m-n-T$ type architecture using the proposed rules. The same fault patterns are used, as shown in Fig. 8, and value of $F_{\text{in}}[i,j]$ is also shown on each PE. Note that PE[2,3] in Fig. 10(b) need not to be deactivated in the case where $T = 2$.

We can also define rules to change switch functions for architectures of $T \geq 3$, and they will show higher reconfiguration performance, but such rules will require information in PEs which are far from the switch, thus becoming more complicated. This seems unrealistic from the viewpoint of keeping hardware overhead to a minimum. The proposed rules can be implemented as simple combinational circuits, as illustrated in Fig. 14, which will be explained later.

As described above, each step of the BC method is performed by sending and receiving the fault information locally. This

```
if F_in[i,j] − F_in[i,r(j)] − 1 then
  SW_1[i,j] := NW;
else if F_in[i,j] − F_in[i,r(j)] = −1 then
  SW_1[i,j] := NE;
else if F_in[i,j] − F_in[i,r(j)] = 0 then
  if (unused[i,j], unused[i,r(j)]) = (0,0) then
    SW_1[i,j] := EW;
  else if (unused[i,j], unused[i,r(j)]) = (1,0) then
    SW_1[i,j] := NW;
  else if (unused[i,j], unused[i,r(j)]) − (0,1) then
    SW_1[i,j] := NE;
  else if (unused[i,j], unused[i,r(j)]) = (1,1) then
    SW_1[i,j] := NC;
  end if
end if
```

(a) Rule for T=1.

```
if F_in[i,j] − F_in[i,r(j)] = 2 then
  SW_1[i,j] := NW, SW_2[i,j] := NW;
else if F_in[i,j] − F_in[i,r(j)] = −2 then
  SW_1[i,j] := NE, SW_2[i,j] := NE;
else if F_in[i,j] − F_in[i,r(j)] = 1 then
  SW_2[i,j] := NW;
  if unused[i,j] = 0 and unused[i − 1,r(j)] = 0 then
    SW_1[i,j] := EW; else SW_1[i,j] := NW;
  end if
else if F_in[i,j] − F_in[i,r(j)] = −1 then
  SW_1[i,j] := NE;
  if unused[i − 1,j] = 0 and unused[i,r(j)] = 0 then
    SW_2[i,j] := EW; else SW_2[i,j] := NE;
  end if
else if F_in[i,j] − F_in[i,r(j)] = 0 then
  if (unused[i,j], unused[i,r(j)]) − (0,0) then
    SW_1[i,j] := EW, SW_2[i,j] := EW;
  else if (unused[i,j], unused[i,r(j)]) = (1,0) then
    SW_1[i,j] := NC, SW_2[i,j] := NW;
  else if (unused[i,j], unused[i,r(j)]) = (0,1) then
    SW_1[i,j] := NE, SW_2[i,j] := NC;
  else if (unused[i,j], unused[i,r(j)]) = (1,1) then
    SW_1[i,j] := NC, SW_2[i,j] := NC;
  end if
end if
```

(b) Rule for T=2.

(a) (b)

Fig. 9. Rules for changing switch functions. (a) Rule for $T = 1$. (b) Rule for $T = 2$.

property makes the hardware implementation of the BC method easier.

IV. Experimental Results

A. Array Yield

To evaluate the effectiveness of the proposed self-reconfiguration method, its array yield is compared with those from previous studies [2], [3], [7], [24]. Monte Carlo simulations were carried out for the BC method using a large number of randomly-generated datasets. Many researchers have assumed fault-free redundant hardware, such as switches and tracks [4]–[8], [11], [14], [16], [18]–[24]. However, as the chip area for switches and tracks becomes larger, it is difficult to ignore the probability of their being faulty [15]. Hence, the influence of faults on switches and tracks are taken into consideration. The assumptions made in this evaluation are as follows.

1) All PEs, switches, and tracks are assumed to be faulty.
2) The links between a PE and a switch, bypass controllers, and port selectors, as depicted in Fig. 1, are assumed to be fault-free.
3) Faults occur randomly. [4]–[8], [11], [14], [16], [18]–[24].

In this evaluation, array yield is defined as

$$\text{Array yield} = Y_{\text{array}} \times Y_{\text{st}}$$

where Y_{arrray} is the probability of obtaining an $m \times n$ logical array from an $M \times N$ physical array, which is evaluated by Monte Carlo simulation, and Y_{st} is the probability that all switches and tracks are fabricated without faults. Y_{st} can be estimated by the total area of switches and tracks [25]. Let W_{PE}, W_{SW}, and W_{track} be the width of a PE, a switch, and a track, respectively, and L_{track} be the length of a track between two switches. For the switches and tracks capable of propagating 32 bits data in both directions, W_{SW}, W_{track}, and L_{track} are assumed to be 600 λ, 200 λ, and W_{PE}, respectively, where λ is assumed to be 0.2 μm [25]. Taking into account the number of all switches and tracks, the total area of switches and tracks (A_{st}) can be calculated easily. Then Y_{st} is given by $e^{-A_{\text{st}}D}$, where D represents average fault density (faults/cm^2), and yield of a PE (Y_{PE}) is given by $e^{-W_{\text{PE}}^2 D}$. We assume $D = 0.2$ as a typical value of today's technology [25].

With the above assumptions, the array yield of our BC method is compared with that of the BS method [7], which also allows column bypass and one-directional rerouting to achieve self-reconfiguration, and with the theoretically efficient methods: Kung's method [2], [3] and Mahapatra's method [24]. Note that both Kung's and Mahapatra's methods search the assignment of spare PEs exhaustively by graph theory; however, they are ultimately inappropriate for self-reconfiguration.

Fig. 11 shows the array yield as a function of W_{PE} when a logical array of size 20×20 is obtained from a physical array of size 22×22; that is, the array yield of a 22-22-20-20-1 type architecture. The array yields of the BC, the BS, and Kung's method are shown in Fig. 11(a) under the condition of fault-free switches and tracks $(A_{\text{st}} = 0)$. Each array yield is the average of 10 000 simulation results. Yield of a PE, Y_{PE}, is also shown in the figure. The BC method achieves almost the same array yield as that of the BS method without using recursive signals. The array yield for the Kung's method is slightly higher than those of the other two methods, since it uses global information and twice as many switches and tracks.

Fig. 11(b) shows the three array yields in the same fashion as in Fig. 11(a), except that A_{st} is considered to evaluate the influence of faulty switches and tracks. The influence of faults on switches and tracks, decreases all yields compared to those in Fig. 11(a). In this case, the yield of the BC method is higher than that of Kung's method, where W_{PE} is smaller than 0.5 cm.

The array yield of the 24-24-20-20-2 type architecture with fault-free switches and tracks is shown in Fig. 12(a). In the case

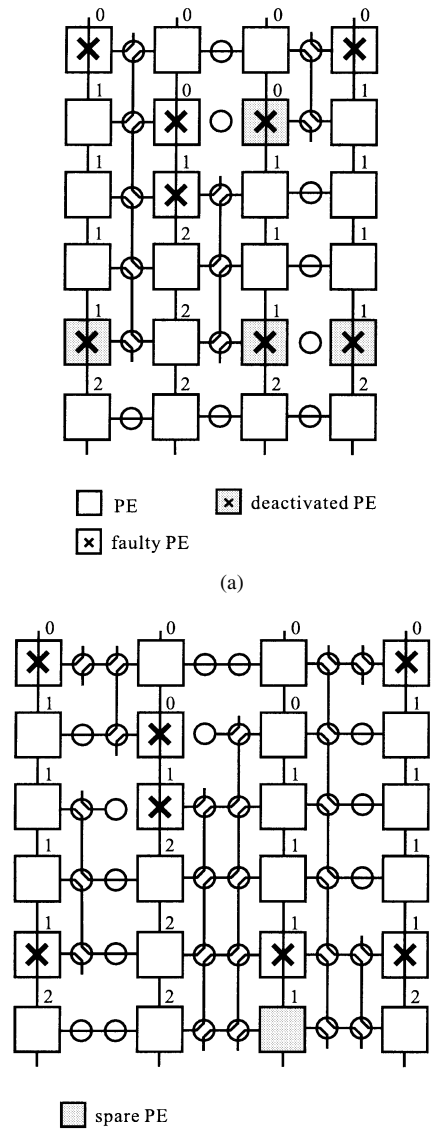


Fig. 10. Reconfiguration of $M-N-m-n-T$ type architecture. (a) $T = 1$. (b) $T = 2$.

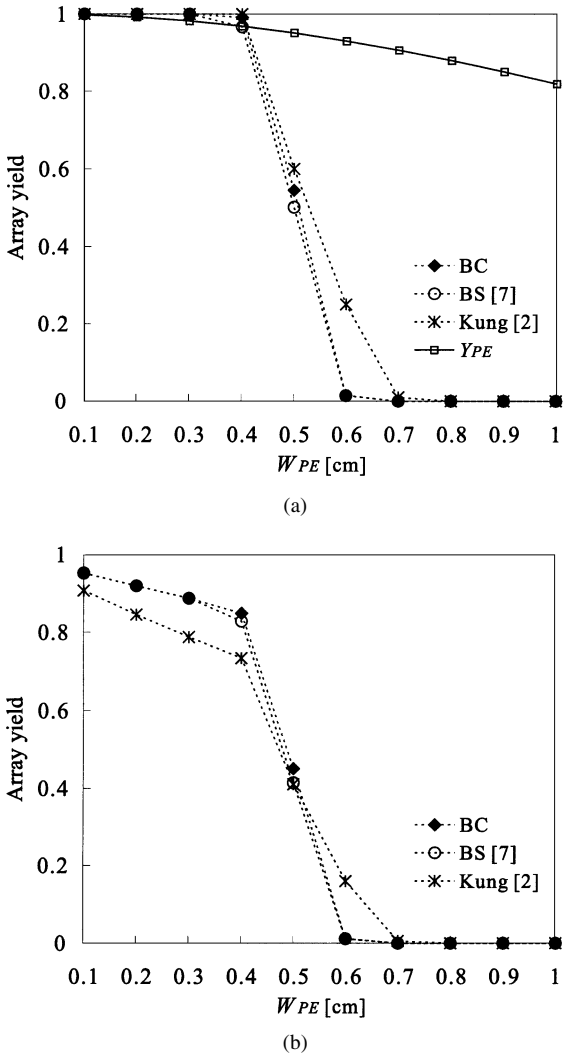


Fig. 11. Array yields of 22-22-20-20-1 type as a function of W_{PE}. (a) Fault-free switches and tracks. (b) Faulty switches and tracks.

where $T = 2$, both the BC and the BS methods achieve better performance than Kung's method, since the array yield from the latter is obtained by a graph theory approximation for $T = 1$. However, the yield of Mahapatra's method is much higher than those of the other two methods. This is because Mahapatra's method requires double-track switches between every two adjacent rows and columns, and switches are allocated at each track intersection point as well as each interconnection between PEs. Furthermore, that method allows efficient rerouting in eight directions. Mahapatra's method utilizes only one spare row or column around an array. Thus, to keep the same number of spare PEs, the yield in Fig. 12(a) is estimated by dividing the original array of size 24×24 into four arrays of size 12×12.

Fig. 12(b) shows the four array yields of the 24-24-20-20-2 type architecture under the condition that switches and tracks may be faulty. In this case, the array yield of each method is much lower than in Fig. 11(b). Especially, the yield of Mahapatra's method decreases significantly due to more than double redundant hardware than that of the BC method. Our BC method is efficient when W_{PE} is smaller than or equal to 0.8 cm.

Since the application of the BC method is not limited to the reconfiguration of regular physical arrays, the array yields of the BC method are evaluated for rectangular physical arrays. Since faulty PEs can be avoided by south-directional rerouting, such rectangular arrays, with more spare rows at the bottom of the array, will produce better performance.

Fig. 13(a) plots the array yields of the BC method in the same fashion as in Fig. 11(b), when a logical array of size 20×20 is obtained from physical arrays of size 23×21, 24×20, and 22×22. Note that almost the same number of nonspare PEs and spare PEs in each array are retained. As the number of rows increases, the array yields show better performance. The array yield of the 24-20-20-20-1 type without column bypass shows the best result among all the rectangular arrays and also achieves better array yield than that of Kung's method in Fig. 11(a).

Fig. 13(b) shows the array yields of rectangular physical arrays where $T = 2$. These results show the same tendencies as Fig. 13(a), although the array yield of the 27-21-20-20-2 shows the best result, better that of Mahapatra's method even when

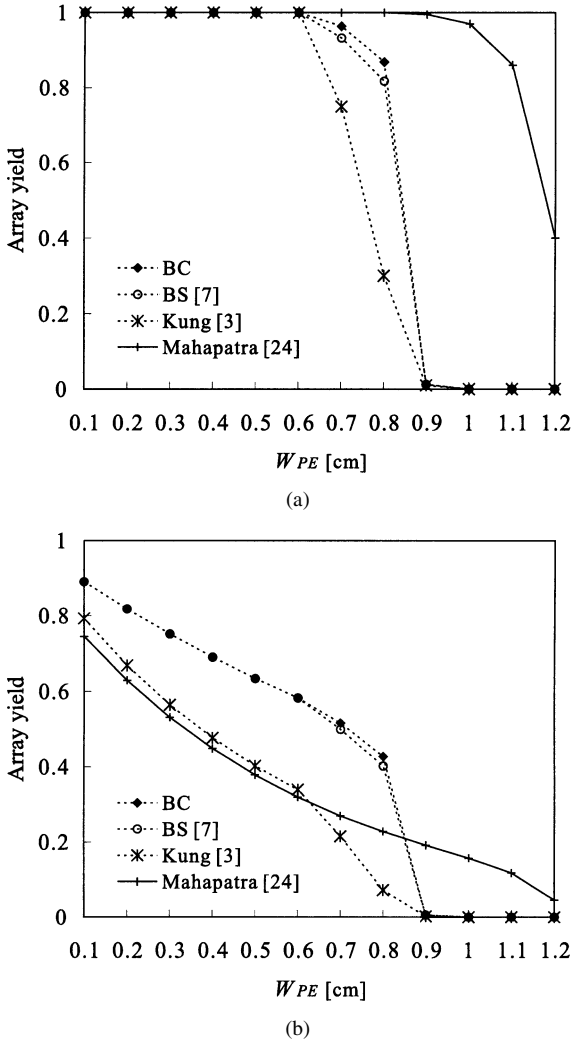


Fig. 12. Array yields of 24-24-20-20-2 type as a function of $W_{\rm PE}$. (a) Fault-free switches and tracks. (b) Faulty switches and tracks.

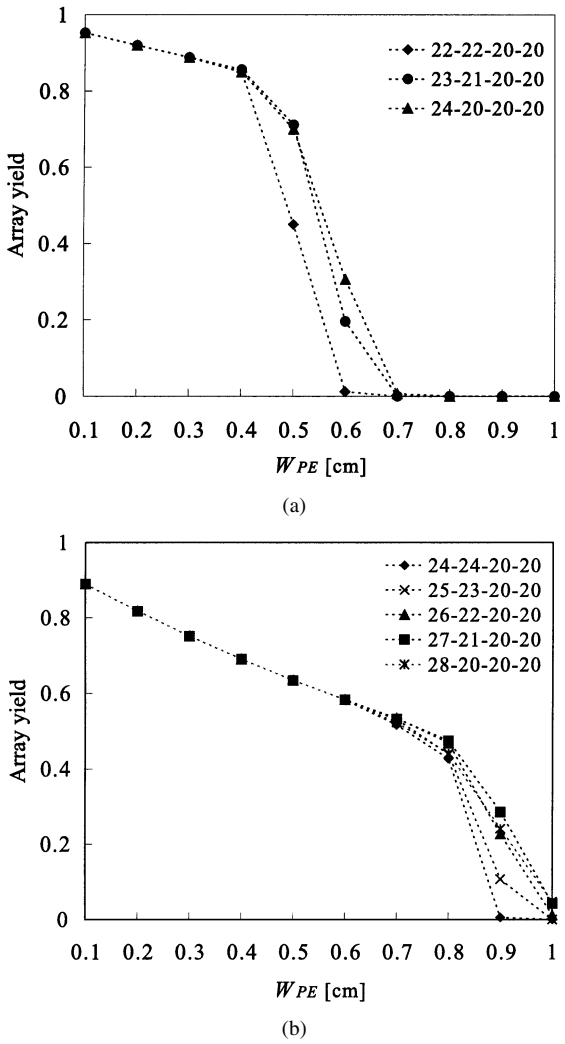


Fig. 13. Array yields for rectangular physical arrays under the condition of faulty switches and tracks. (a) $T = 1$. (b) $T = 2$.

$W_{\rm PE} = 0.9$ cm. These simulations verify that the BC method efficiently utilizes spare PEs if the physical arrays have more spare rows. Note that the self-reconfiguration methods proposed by Takanami *et al.* [14], [16] are approximations of Kung's method. Hence, our method outperforms their method with half the number of switches and tracks.

B. Reconfiguration Time

The reconfiguration time of the BC method is defined as the total number of steps to reconfigure an $m \times n$ logical array from an $M \times N$ physical array. Step 1 of the BC method requires just $M + 2N(N - n)$ time steps, since it takes M steps to count $f_{\rm out}[M, j]$ in all columns and $2N$ steps to bypass one column. From Lemma 1, step 2 takes $M \times N$ time steps and step 3 takes only 1 time step to change all switch functions. Therefore the number of time steps required for the BC method is constant and estimated as follows:

$$T_{\rm BC}(M, N, m, n) = M + 2N(N - n) + MN + 1.$$

Consequently, the time complexity of the BC method is $\mathrm{O}(MN)$.

On the other hand, the time complexity of the approximation method [16] is given as $\mathrm{O}(N^2)$, where an $N \times N$ array is obtained from an $(N + 2) \times (N + 2)$ array. Both methods show almost the same complexity.

V. Prototype of Self-Reconfigurable Mesh Arrays on FPGA

This section discusses the hardware implementation of the BC method using FPGA to verify that the BC method can be embedded into a mesh array. Here, a prototype of self-reconfigurable mesh arrays with single-track switches ($T = 1$) is designed and implemented.

A. Hardware Implementation

1) Design Environment: All components in Fig. 1, namely PEs, switches, bypass controller, port-selector, and interconnections are designed using the following tools and instruments.

1) **Design tool**:
   - MAX+PlusII 9.6 (ALTERA, Inc.);
   - Leonardo Spectrum V1999.1 (MGC[1]).

[1]Mentor Graphics Corporation.

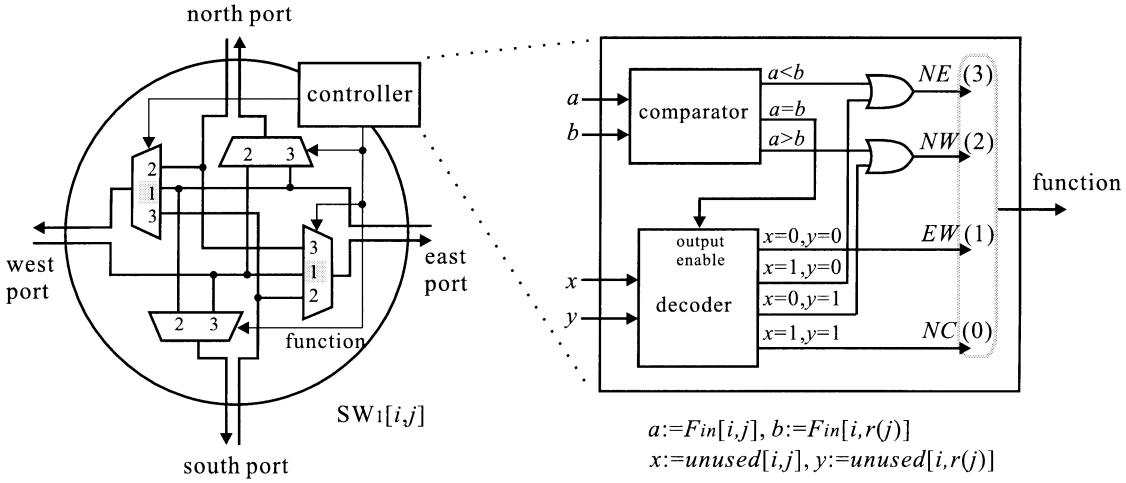


Fig. 14. Switch circuit.

2) **FPGA**: EPF10K250AGC599-3 (ALTERA, Inc.).

3) **System board**:

- MEB200-A250: main board to implement FPGA (MMS[2]);
- MU200-EA10: board for data I/O (MMS).

2) Switch: A switching circuit for the $M-N-m-n-1$ type architecture consists of four multiplexors and a controller, as illustrated in Fig. 14. The switch function is automatically determined by the controller; in other words, the rules shown in Fig. 9 are implemented in this controller. According to the determined function, the actual connections of all ports are established using the multiplexers. For example, if the function is determined as *EW*, then the value 1 is input to all multiplexers to connect the east and west ports to each other.

3) PE: A PE is designed to execute only the functions required to perform the BC method, though the detailed hardware circuits are omitted here. Each PE$[i,j]$ consists of two internal switches, flip-flops to hold $\mathrm{fault}[i,j]$ and $\mathrm{unused}[i,j]$, an accumulator for $f_{\mathrm{out}}[i,j]$, a signal generator to deactivate PE$[i-2,r(j)]$ and PE$[i-2,l(j)]$, a decision maker circuit which indicates whether the PE is deactivated or not, and data selectors when the PE is bypassed. The data selectors have the following important role: suppose that PE$[i,j]$ is bypassed and both PE$[i,l(j)]$ and PE$[i,r(j)]$ are connected to each other. In this case, PE$[i,l(j)]$ requires information about PE$[i,r(j)]$ instead of PE$[i,j]$, both to perform deactivation and to determine switch functions correctly, and vice versa. For example, the information held in PE$[i,r(j)]$ such as $\mathrm{unused}[i,r(j)]$, $f_{\mathrm{in}}[i,r(j)]$, and the signal for deactivation will be supplied to PE$[i,l(j)]$ by the selectors.

4) Bypass Controller: A bypass controller circuit allocated at the bottom of each column is shown in Fig. 15. FF1 in Fig. 15 configures an N-length token ring by connecting to other controllers in order to provide the token. Only the column that gets the token can execute the bypass algorithm. The values $f_{\max}$ and $a_{\max}$ are input from the previous $l(j)$th column and are compared with $f_{\mathrm{out}}[i,j]$ and $a[j]$, respectively. According to the conditions in the bypass algorithm, $f_{\max}$ and $a_{\max}$ may

[2]Mitsubishi Electronic Micro-computer Application Software Co., LTD.

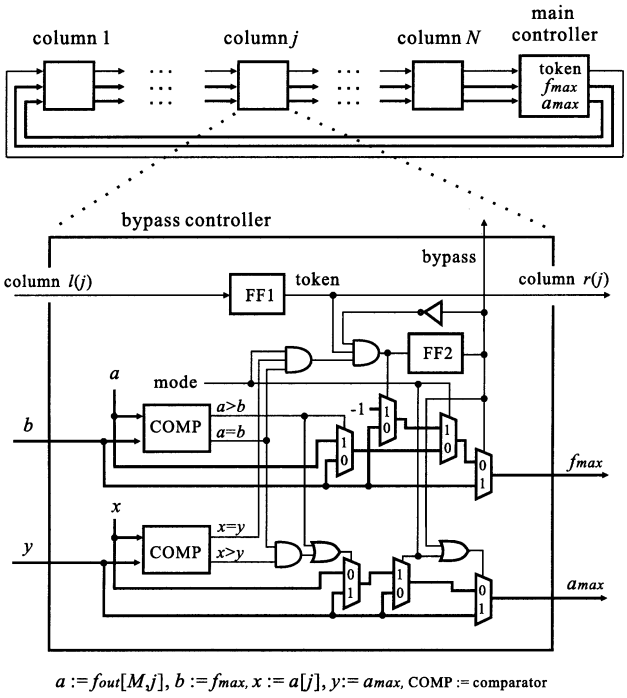
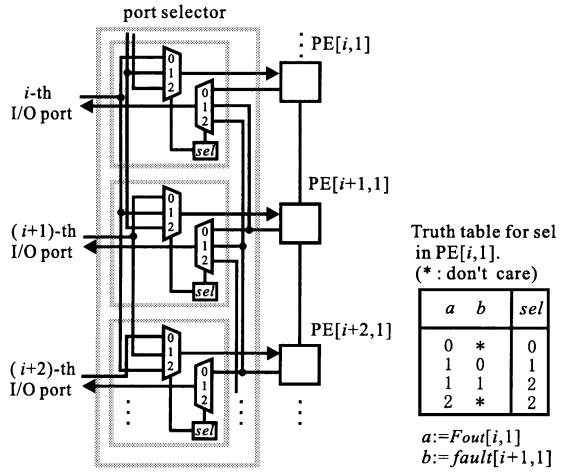


Fig. 15. Bypass controller for a column.

be changed and selected using multiplexers, then they are output to the next $r(j)$th column. The signal named "mode" in Fig. 15 denotes the first search (mode="low") or the second search (mode="high") as presented in Fig. 4. In the second search, if $f_{\mathrm{out}}[i,j]$ and $a[j]$ equal $f_{\max}$ and $a_{\max}$, respectively, then this column, which has not yet been bypassed, is selected to be bypassed. Once the column is bypassed, FF2 shown in Fig. 15 continues to output "high" and $f_{\max}$ and $a_{\max}$ pass through this column.

5) Port Selector: The port selector consists of m connecting circuits to connect m rows on the leftmost and rightmost sides and n connecting circuits to connect n columns on the uppermost and lowest sides with external I/O ports. Fig. 16 shows an example of the circuits which allocate the leftmost side of



a	b	sel
0	*	0
1	0	1
1	1	2
2	*	2

Fig. 16. Port selector.

the array. Each connecting circuit has two multiplexers to select the inputs and outputs. If $(M-m)$ rows are dedicated as spares, PE$[i,1]$ on the leftmost column can be shifted into position PE$[i+M-m,1]$ by the south-directional rerouting. Hence, one multiplexer selects one port among the $(M-m+1)$ ports to connect to the external I/O port. Fig. 16 shows the example circuit when $(M-m)=2$.

6) Prototype of an 8-8-6-6-1 Type Architecture: Using the above components, an 8-8-6-6-1 type architecture has been designed and implemented on an FPGA. It is assumed that each interconnection consists of two bus-lines for inputs and outputs and that they are both 8 bits. Some fault patterns are also embedded into PEs and after reconfiguration we check all the I/O ports, switch functions, and PE states to determine whether or not the reconfiguration has been successful. The expected results are observed according to the embedded fault patterns and used to verify that the proposed self-reconfigurable architecture can work on hardware when the clock frequency is 20 MHz. Since this frequency is the maximum frequency of the FPGA board, our system seems to work at higher clock frequencies. Note that the prototype system requires almost the same number of clocks as time steps T_{BC} shown in Section IV. In the 8-8-6-6-1 type system, the reconfiguration time is about 5.5 μs, where the clock frequency is 20 MHz, and the time does not change according to the fault distributions. Hence, the system is useful for run-time as well as fabrication-time reconfiguration because it requires no host computer to execute the reconfiguration and the time is very short.

B. Discussion of Hardware Overhead

The number of gates needed to implement each component has been accurately estimated using the Leonardo Spectrum design tool with the XCL05U ASIC library. From the design tool, a switch, a PE, a bypass controller allocated at bottom of each column, and a port selector for one I/O are reported to require 490 gates, 1368 gates, 552 gates, and 371 gates, respectively, where each interconnection is 16-bits.

Let $H(M,N,m,n,P)$ be the total number of gates required for the $M-N-m-n-1$ type architecture where P is the number of gates for a PE, which includes actual calculation circuits. Taking into account the numbers of each circuit, the total number of gates required for the $M-N-m-n-1$ type architecture with 16-bits interconnections is computed as follows:

$$H(M,N,m,n,P)=490M(N-1)+(1368+P)MN+552B+317(2m+2n).$$

To evaluate the hardware overhead of the proposed self-reconfigurable architecture, the overhead ratio $O(M,N,m,n,P)$ is defined as follows:

$$O(M,N,m,n,P)=\frac{H(M,N,m,n,0)}{H(M,N,m,n,P)}\times 100\%.$$

TABLE I
NUMBER OF GATES AND OVERHEAD FOR 8-8-6-6-1 TYPE

the number of gates for a PE: P (K gates)	$H(8,8,6,6,P)$ (K gates)	$O(8,8,6,6,P)$ (%)
10	767	14.2
20	1407	8.3
30	2047	5.8
40	2687	4.5
50	3327	3.7

Table I shows $H(M,N,m,n,P)$ and $O(M,N,m,n,P)$ for the 8-8-6-6-1 type prototype system where P ranges from 10-K gates to 50-K gates. When P is 10-K gates, the overhead is about 14%. In this case, the redundant circuits are subject to faults and will decrease the system yield. When P is 50-K gates, the overhead is at a very small percentage, 3.7%. In this case, the redundant circuits are less vulnerable to faults.

VI. CONCLUSION

As the number of PEs in a massively parallel system increases, it has become necessary to develop self-reconfigurable systems which can automatically reconfigure partially faulty systems. We have proposed a self-reconfiguration method for mesh arrays and demonstrated its implementation using an FPGA. The proposed switching mechanism, which can determine the desired switch functions automatically using the states of neighboring PEs, makes the hardware implementation easier. Simulated results show that the proposed method employing simple schemes achieves higher array yields than those of previous methods with only half the number of switches and tracks, especially when the physical arrays have more rows of spare PEs than columns. Finally, the prototype system of self-reconfigurable mesh arrays is implemented using an FPGA and the correct behavior of the reconfiguration was verified. The system is useful for both run-time and fabrication-time reconfiguration because it requires no host computer to execute the reconfiguration and the reconfiguration time is very short. The overhead for redundant circuits such as switches and control circuits is also evaluated and shown to be less than 4% where the number of gates for a PE is 50-K gates.

REFERENCES

[1] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. New York: McGraw-Hill, Dec. 1992.

[2] S. Y. Kung, S. N. Jean, and C. W. Chen, "Fault-tolerant array processors using single track switches," *IEEE Trans. Comput.*, vol. 38, pp. 501–514, Apr. 1989.

[3] J. S. N. Jean, H. C. Fu, and S. Y. Kung, "Yield enhancement for WSI array processors using two-and-half-track switches," in *Proc. Int. Conf. Wafer Scale Integration*, Jan. 1990, pp. 243–250.

[4] V. P. Roychowdhury, J. Bruck, and T. Kailath, "Efficient algorithms for reconstruction in VLSI/WSI array," *IEEE Trans. Comput.*, vol. 39, pp. 480–489, Apr. 1989.

[5] T. A. Varvarigou, V. P. Roychowdhury, and T. Kailath, "A polynomial time algorithm for reconfiguring multiple-track models," *IEEE Trans. Comput.*, vol. 42, pp. 385–395, Apr. 1993.

[6] ——, "Reconfiguring processor arrays using multiple-track models: The 3-track-1-spare-approach," *IEEE Trans. Comput.*, vol. 42, pp. 1281–1293, Nov. 1993.

[7] I. Numata and S. Horiguchi, "A self-reconfiguration architecture for mesh arrays," in *Proc. Int. Workshop Defect and Fault Tolerance in VLSI Systems*, Oct. 1994, pp. 212–220.

[8] ——, "Wafer-scale integration implementation of mesh-connected multiprocessor systems," *Syst. Comput. Jpn.*, vol. 26, no. 1, pp. 1–10, Jan. 1995.

[9] A. D. Singh, "Interstitial redundancy: An area efficient fault-tolerance scheme for large area VLSI processor arrays," *IEEE Trans. Comput.*, vol. 37, pp. 1398–1410, Nov. 1988.

[10] M. Wang, M. Cutler, and S. Y. H. Su, "Reconfiguration of VLSI/WSI mesh array processors with two-level redundancy," *IEEE Trans. Comput.*, vol. 38, pp. 547–554, Apr. 1989.

[11] D. Bhatia, R. Rajagopalan, and S. Katkoori, "Hierarchical reconfiguration of VLSI/WSI arrays," in *Proc. Int. Conf. VLSI Design*, Jan. 1994, pp. 349–352.

[12] F. Lombardi and W. K. Huang, "Approaches for repair of VLSI/WSI DRAM's by row/column deletion," in *IEEE Fault-Tolerant Computing Symp.*, June 1988, pp. 342–347.

[13] M. Chean and J. A. B. Fortes, "The Full-Use-of-Suitable Spares (FUSS) approach to hardware reconfiguration for fault-tolerant processor arrays," *IEEE Trans. Comput.*, vol. 39, pp. 564–571, Apr. 1990.

[14] I. Takanami, K. Kurata, and T. Watanabe, "A neural algorithm for reconstructing mesh-connected processor arrays using single-track switches," in *Proc. Int. Conf. Wafer Scale Integration*, 1995, pp. 501–514.

[15] Y. Y. Chen, S. J. Upadhyaya, and C. H. Cheng, "A comprehensive reconfiguration scheme for fault-tolerant VLSI/WSI array processors," *IEEE Trans. Comput.*, vol. 46, pp. 1363–1371, Dec. 1997.

[16] T. Horita and I. Takanami, "An efficient method for reconfiguring the 1 1/2-track switch mesh array," *IEICE Trans. Inform. Syst.*, vol. E-82D, pp. 1545–1553, Dec. 1999.

[17] ——, "An FPGA implementation of a self-reconfigurable system for the 1 1/2-track switch 2-D mesh array with PE faults," *IEICE Trans. Inform. Syst.*, vol. E-83D, no. 8, pp. 1701–1705, Aug. 2000.

[18] ——, "Fault-tolerant processor arrays based on the 1 1/2-track switches with flexible spare distributions," *IEEE Trans. Comput.*, vol. 49, pp. 542–552, June 2000.

[19] J. H. Kim and P. K. Rhee, "The rule-based approach to reconfiguration of 2-D processor arrays," *IEEE Trans. Comput.*, vol. 42, pp. 1403–1408, Nov. 1993.

[20] M. D. Smith and P. Mazumder, "Generation of minimal vertex covers for row/column allocation in self-repairable arrays," *IEEE Trans. Comput.*, vol. 45, pp. 109–115, Jan. 1996.

[21] S. Y. Kuo and I. Y. Chen, "Efficient reconfiguration algorithms for degradable VLSI/WSI arrays," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 1289–1300, Oct. 1992.

[22] C. P. Low and H. W. Leong, "On the reconfiguration of degradable VLSI/WSI arrays," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 1213–1221, Oct. 1997.

[23] C. P. Low, "An efficient algorithm for degradable VLSI/WSI arrays," *IEEE Trans. Comput.*, vol. 49, pp. 553–559, June 2000.

[24] N. R. Mahapatra and S. Dutt, "Hardware-efficient and highly reconfigurable 4- and 2-track fault-tolerant design for mesh-connected arrays," *J. Parallel Distrib. Comput.*, vol. 61, no. 10, pp. 1391–1411, Oct. 2001.

[25] B. M. Maziarz and V. K. Jain, "Yield estimations for the TESH multicomputer network," in *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems*, Nov. 2002, pp. 20–28.

Masaru Fukushi received the M.S. degree from Hirosaki University, Japan, in 1997 and the Ph.D degree in information science from the Graduate School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, in 2002.

He is currently a Research Associate with the Graduate School of Information Science, JAIST. His research interests are fault-tolerant multiprocessor systems and parallel image processing.

Susumu Horiguchi (SM'95) received the M.E and D.E. degrees from Tohoku University, Japan, in 1978 and 1981, respectively.

He was with the faculty of the Department of Information Science, Tohoku University, from 1981 to 1992. He was a Visiting Scientist with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, from 1986 to 1987, and a Visiting Professor with The Center for Advanced Studies, the University of Southwestern Louisiana, Lafayette, and the Department of Computer Science, Texas A&M University, College Station, during the summers of 1994 and 1997. Since 1992, he has been a Full Professor with the Graduate School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, where he holds the Chair of Multi-Media Integral Systems. His research interests have been mainly concerned with interconnection networks, parallel computing algorithms, massively parallel processing, parallel computer architectures, VLSI/WSI architectures, and multimedia integral systems.

Prof. Horiguchi is a member of the IEICE, IPS, and IASTED. He has been involved in organizing many international workshops, symposia, and conferences sponsored by the IEEE, IEICE, and IPS.